Apple Assembly Line

$1.80

Volume 4 -- Issue 6 March, 1984

In This Issue...

For some time now we have been selling our S-C Word Processor, complete with all source code on disk. We hoped that some users would send us their improvements, and sure enough they have. Bob Gardner recently sent us a bunch, and that motivated me to go back over the package.

The disk now includes both II/IIPlus and //e versions. The //e version allows an 80-column preview (still only 40-column during edit mode). I added titles and page numbers, a single sheet mode, and more. Even with all the new features, the new object code is a little shorter than the old, leaving even more room for your own modifications and enhancements. I improved the internal documentation. The "manual-ette" is now 10 pages rather than 6. A small tutorial file helps you get started.

The price is still only $50. Owners of the old version can get a new copy for only $5.

Fast Garbage Collection................Col. Paul Shetler, MD

Honolulu, Hawaii

When Applesoft programs manipulate strings, memory gradually fills up with little bits and pieces of old strings. Eventually this space needs to be recovered so the program can continue. The process of searching through all the still active strings, moving them back to the top of free memory, and making the remaining space available again is called "garbage collection".

Applesoft will automatically collect garbage when memory fills up. However, the garbage collector in the Applesoft ROMs is pitifully slow. Worse yet, the time to collect is proportional to the square of the number of strings in use. That is, if you have 100 active strings it will take four times as long to collect garbage as if you had only 50 active strings.

Cornelis Bongers, of Erasmus University in Rotterdam, Netherlands, published a brilliant Garbage Collector for Applesoft strings in Micro, August 1982. The speed of his program, when compared to the one residing in ROM, is incredible. And the time is directly proportional to the number of strings, rather than the square of the number of strings. The only problem with his program is that it belongs to the magazine that published it. Or worse yet, it is tied to a program called Ampersoft, marketed by Microsparc (publishers of Nibble magazine) for $50. When I asked them about a license, they wanted big bucks.

So, I decided to write my own garbage collector, based on the ideas behind Cornelis Bongers' program. And then I further decided to make it available to all readers of Apple Assembly Line, where I myself have received so much help.

There are several catches. Normal Applesoft programs save all string data with the high-order bit of each byte zero (positive ASCII). Further, normal Applesoft programs never allow more than one string variable to point to the same exact memory copy of the string. The method of garbage collection my program uses (Bongers' method) DEPENDS on these constraints. If either is not true, LOOK OUT! Of course, if your Applesoft programs are normal, you need have no fear. Only if you are doing exotic things with your own machine language appendages to Applesoft might these constraints be violated.

The basic concept is fairly simple. Applesoft uses descriptors to point to the string in the string pool. The descriptor consists of three bytes -- the length, and the address of the characters in the string pool.

Strings build down from the top of memory (HIMEM) and the descriptors build up from the end of the program in the variable space. Since a new value assigned to a string is added to the bottom of the string pool, the pool is soon full of "garbage".

Applesoft frees the garbage one string at a time. This

S-C Macro Assembler Version 1.0.....................................$80.00
S-C Macro Assembler Version 1.1 Update..............................$12.50
Full Screen Editor for S-C Macro Assembler..........................$49.00
    Includes complete source code.
S-C Cross Reference Utility...$20.00
S-C Cross Reference Utility with Complete Source Code...............$50.00
DISASM Dis-Assembler (RAK-Ware).....................................$30.00
Quick-Trace (Anthro-Digital)......................(reg. $50.00) $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00) $45.00

S-C Word Processor (the one we use!)................................$50.00
    With fully commented source code.
Applesoft Source Code on Disk.......................................$50.00
    Very heavily commented. Requires Applesoft and S-C Assembler.
ES-CAPE: Extended S-C Applesoft Program Editor......................$60.00

AAL Quarterly Disks...each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	

Double Precision Floating Point for Applesoft.......................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)......................(reg. $75.00) $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00) $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95) $60.00
FLASH! Integer BASIC Compiler (Laumer Research).....................$79.00
Fontrix (Data Transforms)... $75.00
Aztec C Compiler System (Manx Software)............(reg. $199.00) $180.00
IACcalc Spreadsheet Program.........................(reg. $84.95) $75.00
        The one we use every day. It's better than Visicalc!

Blank Diskettes...................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each.................10 for $6.00
Diskette Mailing Protectors.........................10-99: 40 cents each
                                              100 or more: 25 cents each
   Cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers.................................. 5 cents each
ZIF Game Socket Extender..$20.00

Buffered Grappler+ Interface and 16K Buffer............($239.00) $200.00
quikLoader EPROM Card...................................($179.50) $170.00

Books, Books, Books..........................compare our discount prices!
    "The Apple][Circuit Description", Gayler...........($22.95) $21.00
    "Understanding the Apple II", Sather.................($22.95) $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95) $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95) $7.50
    "Beneath Apple DOS", Worth & Lechner.................($19.95) $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95) $36.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95) $18.00
    "What's Where in the Apple", Second Edition..........($24.95) $23.00
    "What's Where Guide" (updates first edition)..........($9.95) $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95) $18.00
    "6502 Subroutines", Leventhal........................($17.95) $17.00

Add $1.50 per book for US shipping. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
***   (214) 324-2050   ***
*** We accept Master Card, VISA and American Express ***

n-square method takes forever, when there are large string arrays. Bongers introduced the idea of marking active strings in the pool by setting the third byte in the string to a negative ASCII value, then storing the location of the descriptor in the first two bytes. The first two bytes of the string are saved safely in the address of field of the descriptor. The address previously in the address field will be changed anyway after all the strings are moved up in memory.

Another pass through the string pool moves all active strings as high in memory as they can go, retrieves the first two characters from storage in the descriptor, and points it to the new string location.

Since three bytes are used in the active strings, one and two character strings require different treatment. On the first pass through the variable space, the characters pointed to by the 'short' descriptors are stored in the length and, if len=2, the low address byte of the descriptor. The short descriptor is flagged with one or more "FF"'s, since no string can have an address greater than $FF00.

If short strings are found on the first pass, a third pass returns them to the string pool and points the descriptors to their new location.

Short strings do slow collection a little, however, the number of passes is proportional to the number of strings, and not the number squared.

Bongers' program was driven by calls via the &-statement. Mine differs in that it invoked with the USR function. Although it is easily converted to an ampersand routine, I wrote it using the USR function to provide fast garbage collection with Hayden's compiler (which also uses string descriptors and a string pool). The compiler allows USR functions, but makes & difficult. Another reason is to investigate some uses for USR.

USR(#) converts '#' to a floating point value in the FAC (floating point accumulator) and then jumps via $0A to the address pointed to in $0B, $0C. The results of the machine language subroutine can be returned in the FAC. The USR function, floating point calls, and addresses are described in Apple's BASIC REFERENCE MANUAL FOR APPLESOFT (Product #A2L0006).

The USR argument for my garbage collector requires a number in the range of +32767 to -32767. If the number is negative, the string pool is checked for negative ASCII. If any such characters are found, USR(-1) will return a value of 0, and no garbage collection will be attempted. If no negative ASCII characters are found, garbage collection will proceed. In this case USR(-1) returns the number of bytes of free space after collection.

If the USR argument is zero, for example K = USR(0), then collection is forced and USR will return the amount of free space. This is slightly faster than calling with USR(-1),

because the preliminary scan for negative ASCII bytes is skipped. But USR(-1) is safer, if you are not sure.

If you use a positive argument N in the USR function, then no garbage collection will be performed unless there is less than 256*N bytes of free space left. Whether or not collection is performed, USR will tell you how much free space is left.

Only the lower five bits of the USR argument are tested. This means that USR(32) is the same as USR(0), USR(33) is the same as USR(1), and so on.

I have shown the program as residing at $9400, but of course you may re-assemble it for any favorite place.

The following Applesoft program makes a lot of garbage, and sees to the collection of it using my garbage collector. If the call to the USR function in line 245 left out, the program dies for 47 seconds while Applesoft does its own garbage collection. With the USR call as shown, the delay is less than one second.

```
100  HIMEM: 37888: REM  $9400
110  PRINT  CHR$ (4)"BLOAD B.FAST GARBAGE COLLECTOR"
120  POKE 10,76: POKE 11,0: POKE 12,148
200 N = 25: DIM A$(N,N),B$(N,N)
210  FOR I = 1 TO N: FOR J = 1 TO N
220 A$(I,J) = "X":B$(I,J) = "Y": NEXT : NEXT
230  FOR I = 1 TO N
240 F = ( PEEK (112) * 256 +  PEEK (111)) - ( PEEK (110) * 256 +  PEEK (1
    09)): PRINT F" ";
245 L =  USR (2)
250  FOR J = 1 TO N
260  FOR K = 1 TO 10:A$(I,J) = A$(I,J) + "*": NEXT
270  PRINT "*";: NEXT
280  PRINT : NEXT
```

```
*SAVE S.FAST GARBAGE COLLECTOR
*--------------------------------
*   FAST GARBAGE COLLECTOR
*--------------------------------
*   BY COL. PAUL SHETLER, MD
*   INSPIRED BY CORNELIS BONGERS
*--------------------------------
*
*   CALL FROM APPLESOFT WITH K=USR(N)
*
*   IF N=0, THEN COLLECTION FORCED
*
*   IF N<0, THEN POOL CHECKED FOR NEG ASCII.
*          IF NO NEG ASCII, THEN GC FORCED
*          IF NEG ASCII FOUND, THEN
*               SET USER(#)=0 AND QUIT.
*
*   IF N>0, THEN COLLECTION PERFORMED ONLY IF
*         LESS THAN N*256 BYTES OF FREE
*         SPACE LEFT.
*--------------------------------
*      THE APPLESOFT PROGRAM MUST INLCUDE
*      THE FOLLOWING STATEMENTS TO SET UP
*      THIS GARBAGE COLLECTOR:
*
*      100 HIMEM:37888:REM$9400
*      110 PRINT CHR$(4)"BLOAD B.FAST GARBAGE COLL
*          ECTOR"
*      120 POKE 10,76 : POKE 11,0 : POKE 12,148
```

```
                *-------------------------------
                * EQUATES FOR GARBAGE COLLECTION
                *-------------------------------
0006-           SHORT.FLAG          .EQ $06
0007-           STRING.LENGTH       .EQ $07
0019-           INDEX               .EQ $19
001B-           OFFSET              .EQ $1B
001D-           ARRAY.END           .EQ $1D
                *-------------------------------
                *      USER(#)  EQUATES
                *-------------------------------
00A0-           FACMO               .EQ $A0
00A1-           FACLO               .EQ $A1
E10C-           AYINT               .EQ $E10C
E2F2-           GIVAYF              .EQ $E2F2
                *-------------------------------
                *      STANDARD EQUATES
                *-------------------------------
009B-           LOWTR               .EQ $9B
0008-           FORPNT              .EQ $08
006D-           STREND              .EQ $6D
0069-           VARTAB              .EQ $69
0071-           FRESPC              .EQ $71
006F-           FRETOP              .EQ $6F
0073-           MEMSIZE             .EQ $73
006B-           ARYTAB              .EQ $6B
                *-------------------------------
                        .OR $9400
                        .TF B.FAST GARBAGE COLLECTOR
                *-------------------------------
                USR.GARBAGE.COLLECTOR
9400- 20 0C E1          JSR AYINT     CONVERT USR ARGUMENT TO INTEGER
                *                     WITH HIBYTE IN FACMO, LO IN FACLO
9403- A5 A0             LDA FACMO     IS # MINUS?
9405- 30 1A             BMI .3        ...NEED TO CHECK FOR NEG ASCII
9407- A5 A1             LDA FACLO
9409- 29 1F             AND #$1F      8136 BYTES
940B- F0 1F             BEQ .4        ...IF =0 THEN FORCED COLLECTION
940D- 18                CLC
940E- 65 6E             ADC STREND+1
9410- C5 70             CMP FRETOP+1
9412- B0 18             BCS .4        ...NEED TO COLLECT NOW
                *---CALC FREE SPACE-------------
9414- 38        .1      SEC
9415- A5 6F             LDA FRETOP
9417- E5 6D             SBC STREND
9419- A8                TAY           LO BYTE
941A- A5 70             LDA FRETOP+1
941C- E5 6E             SBC STREND+1
                *---FLOAT (AY) FOR USR RESULT----
941E- 4C F2 E2  .2      JMP GIVAYF    FLOAT (AY) AND RETURN
                *---CHECK POOL FOR NEG ASCII-----
9421- 20 34 95  .3      JSR SET.STRING.POOL.STROLL
9424- 20 3D 95          JSR FIND.NEXT.NEG.BYTE.IN.POOL
9427- A9 00             LDA #0        PREPARE ZERO IN CASE NEG ASCII
9429- A8                TAY
942A- B0 F2             BCS .2        ...FOUND SOME NEG ASCII
                *---COLLECT GARBAGE NOW----------
942C- 20 4C 94  .4      JSR MARK.ALL.ACTIVE.STRINGS
942F- 20 9D 94          JSR RAISE.ALL.ACTIVE.STRINGS
                *---FINAL CLEAN UP--------------
9432- A5 9B             LDA LOWTR     STORE NEW BOTTOM OF STRING POOL
9434- 85 71             STA FRESPC
9436- A5 9C             LDA LOWTR+1
9438- 85 72             STA FRESPC+1
943A- A5 06             LDA SHORT.FLAG    NEED TO FIX SHORT STRINGS?
943C- F0 03             BEQ .5            ...NO, NOT ANY SHORT ONES
943E- 20 F2 94          JSR FIX.SHORT.STRINGS
9441- A5 71     .5      LDA FRESPC    SET FRETOP TO TOP OF FREE SPACE
9443- 85 6F             STA FRETOP
9445- A5 72             LDA FRESPC+1
9447- 85 70             STA FRETOP+1
9449- 4C 14 94          JMP .1
                *-------------------------------
                *  MARK ACTIVE STRINGS WITH NEG BYTE
                *-------------------------------
                MARK.ALL.ACTIVE.STRINGS
944C- A9 00             LDA #0        FLAG->NONE
944E- 85 06             STA SHORT.FLAG
9450- 20 65 95          JSR INITIATE.SEARCH
```

```
9453- 20 72 95  .1      JSR FIND.NEXT.STRING.VARIABLE
9456- B0 44             BCS .5       ...NO MORE VARIABLES
9458- A0 00             LDY #0       POINT AT LENGTH BYTE OF DESC.
945A- B1 9B             LDA (LOWTR),Y
945C- F0 F5             BEQ .1       STRING LEN =0
                *---CHECK LOCATION OF STRING-----
945E- AA                TAX          SAVE STRLEN IN X-REG
945F- C8                INY          IF STRING DATA INSIDE PROGRAM,
9460- B1 9B             LDA (LOWTR),Y     THEN NO NEED TO FIDDLE
9462- 85 08             STA FORPNT        WITH IT FURTHER.
9464- C5 69             CMP VARTAB
9466- C8                INY
9467- B1 9B             LDA (LOWTR),Y
9469- 85 09             STA FORPNT+1
946B- E5 6A             SBC VARTAB+1      IN PROGRAM?
946D- 90 E4             BCC .1            ...YES, SO PASS
                *---CHECK FOR SHORT STRING-------
946F- E0 03             CPX #3       IF 1 OR 2, SPECIAL TREATMENT
9471- B0 15             BCS .3       ...LONG STRING
                *---SHORT STRING HANDLER---------
9473- 86 06             STX SHORT.FLAG    NON-ZERO TO FLAG
9475- A9 FF             LDA #$FF
9477- 91 9B             STA (LOWTR),Y     MARKER IN 3RD DESC. BYTE
9479- 88                DEY               POINT AT 2ND DESC. BYTE
947A- CA                DEX               CHECK LENGTH
947B- F0 02             BEQ .2            LEN=1, PUT $FF IN 2ND BYTE
947D- B1 08             LDA (FORPNT),Y    LEN=2, SAVE CHAR IN 2ND BYTE
947F- 91 9B     .2      STA (LOWTR),Y
9481- 88                DEY               POINT AT 1ST BYTE OF DESC.
9482- B1 08             LDA (FORPNT),Y    MOVE FIRST BYTE OF STRING
9484- 91 9B             STA (LOWTR),Y        TO DESC.
9486- 10 CB             BPL .1       ALWAYS
                *---LONG STRING HANDLER----------
9488- B1 08     .3      LDA (FORPNT),Y    MARK FIRST BYTE OF STRING
948A- 09 80             ORA #$80          MAKE NEG ASCII
948C- 91 08     .4      STA (FORPNT),Y
948E- 88                DEY               BACK UP TOWARD BEG. OF DATA
948F- 30 C2             BMI .1            ...FINISHED MARKING THIS
9491- B1 08             LDA (FORPNT),Y    SAVE STRING CHAR IN DESC.
9493- C8                INY
9494- 91 9B             STA (LOWTR),Y     IN LAST 2 BYTES
9496- 88                DEY          OF DESCRIPTOR
9497- B9 9B 00          LDA LOWTR,Y  SAVE ADDR INSIDE STRING
949A- B0 F0             BCS .4       ALWAYS SET
                *---FINISHED MARKING STRINGS-----
949C- 60        .5      RTS
                *--------------------------------
                *   MOVE THE STRINGS AS HIGH AS POSSIBLE
                *--------------------------------
                RAISE.ALL.ACTIVE.STRINGS
949D- 20 34 95          JSR SET.STRING.POOL.STROLL
94A0- 86 9C             STX LOWTR+1  STARTS AT TOP
94A2- 85 9B             STA LOWTR    OF STRNG POOL
94A4- 20 3D 95  .1      JSR FIND.NEXT.NEG.BYTE.IN.POOL
                *--------------------------------
                * CARRY CLEAR ON RETURN WHEN THRU
                *--------------------------------
94A7- 90 48             BCC .4       ...NO MORE STRINGS IN POOL
94A9- A0 00             LDY #0
94AB- 29 7F             AND #$7F
94AD- 91 71             STA (FRESPC),Y
                *--------------------------------
                * RESTORE STRING POOL TO POS ASC
                * THEN RESET POINTERS
                *--------------------------------
94AF- 38                SEC
94B0- A5 71             LDA FRESPC   RECOVER ADDR.
94B2- E9 02             SBC #2       OF DESCRIPTOR
94B4- 85 71             STA FRESPC   FROM THE STR
94B6- B0 02             BCS .2       ...NO BORROW
94B8- C6 72             DEC FRESPC+1
94BA- B1 71     .2      LDA (FRESPC),Y
94BC- 85 08             STA FORPNT   AND PUT IT
94BE- C8                INY          IN FORPNT
94BF- B1 71             LDA (FRESPC),Y
94C1- 85 09             STA FORPNT+1
94C3- C8                INY          Y=2
94C4- B1 08             LDA (FORPNT),Y
```

```
                  *-------------------------------
                  * RESTORE STRING BY RETURNING
                  * THE FIRST TWO BYTES WHICH WERE
                  * STORED IN THE DESCRIPTOR.
                  *
                  * THEN POINT DESCRIPTOR TO THE
                  * NEW, CORRECT STRING POSITION.
                  *-------------------------------
94C6- 88                    DEY
94C7- 91 71                 STA (FRESPC),Y
94C9- B1 08                 LDA (FORPNT),Y
94CB- 88                    DEY         Y=0
94CC- 91 71                 STA (FRESPC),Y
94CE- B1 08                 LDA (FORPNT),Y
94D0- 85 07                 STA STRING.LENGTH
94D2- 38                    SEC
94D3- A5 9B                 LDA LOWTR
                  *-------------------------------
                  * POINT LOWTR & STRING DESCRIPTOR
                  * TO BOTTOM OF NEW STRING POOL.
                  *
                  * LOWTR HOLDS THE MOVING BOTTOM
                  * OF THE STRING POOL.
                  *-------------------------------
94D5- E5 07                 SBC STRING.LENGTH
94D7- 85 9B                 STA LOWTR
94D9- C8                    INY
94DA- 91 08                 STA (FORPNT),Y
94DC- A5 9C                 LDA LOWTR+1
94DE- E9 00                 SBC #0
94E0- 85 9C                 STA LOWTR+1
94E2- C8                    INY
94E3- 91 08                 STA (FORPNT),Y
                  *-------------------------------
                  * NOW MOVE THE STRING TO ITS
                  * NEW ADDRESS.
                  *-------------------------------
94E5- A4 07                 LDY STRING.LENGTH
94E7- 88                    DEY
94E8- B1 71       .3        LDA (FRESPC),Y
94EA- 91 9B                 STA (LOWTR),Y
94EC- 88                    DEY
94ED- 10 F9                 BPL .3
94EF- 30 B3                 BMI .1          ...ALWAYS
                  *---FINISHED MOVING STRINGS------
94F1- 60          .4        RTS
                  *-------------------------------
                  *    RESTORE NORMAL CONFIGURATION OF PNTR AND DATA
                  *        FOR STRINGS OF 1 OR 2 CHARACTERS
                  *
                  *        SCAN THRU VARIABLE SPACE AGAIN:
                  *            DESCRIPTORS OF STRINGS MARKED WITH $FF
                  *            CONTAIN THE CHAR(S) TO RESTORE TO POOL.
                  *
                  *        FRESPC POINTS AT BOTTOM OF POOL
                  *        LOWTR POINTS AT DESCRIPTORS
                  *-------------------------------
                  FIX.SHORT.STRINGS
94F2- 20 65 95              JSR INITIATE.SEARCH
94F5- 20 72 95    .1        JSR FIND.NEXT.STRING.VARIABLE
94F8- B0 39                 BCS .5          ...FINISHED!
94FA- A0 02                 LDY #2          POINT AT 3RD BYTE, 2ND OF ADDR
94FC- 84 07                 STY STRING.LENGTH
94FE- B1 9B                 LDA (LOWTR),Y     IF 3RD BYTE =$FF, SHORTY.
9500- C9 FF                 CMP #$FF        A SHORTY?
9502- D0 F1                 BNE .1          ...NO, KEEP SCANNING VARIABLES
9504- 88                    DEY             ...YES, POINT AT 2ND BYTE
9505- B1 9B                 LDA (LOWTR),Y     IF 2ND BYTE ALSO $FF,
9507- 48                    PHA               THEN LEN=1
9508- 10 02                 BPL .2          ...NOT $FF, ITS A STR CHAR
950A- C6 07                 DEC STRING.LENGTH
950C- 88          .2        DEY             POINT AT 1ST BYTE OF DESCRIPTOR
950D- B1 9B                 LDA (LOWTR),Y     GET 1ST ASC CHAR OF STRING
950F- 48                    PHA               SAVE ON STACK
                  *---CALC PLACE IN POOL FOR DATA--
9510- 38                    SEC
9511- A5 71                 LDA FRESPC      REPOINT FRESPC
9513- E5 07                 SBC STRING.LENGTH
9515- 85 71                 STA FRESPC
9517- B0 02                 BCS .3
9519- C6 72                 DEC FRESPC+1
```

```
            *---RESTORE LENGTH TO DESC.------
951B- A5 07 .3     LDA STRING.LENGTH
951D- 91 9B        STA (LOWTR),Y
            *---STORE CHARS INTO POOL--------
            *--AND ADDR INTO DESCRIPTOR------
951F- 68           PLA          FIRST CHAR
9520- 91 71        STA (FRESPC),Y
9522- C8           INY
9523- A5 71        LDA FRESPC   LOBYTE OF ADDR
9525- 91 9B        STA (LOWTR),Y
9527- 68           PLA          2ND CHAR
9528- 30 02        BMI .4       ...IT IS $FF, ONLY 1 CHAR
952A- 91 71        STA (FRESPC),Y
952C- C8    .4     INY
952D- A5 72        LDA FRESPC+1 HIBYTE OF ADDR
952F- 91 9B        STA (LOWTR),Y
9531- D0 C2        BNE .1       ALWAYS
            *---ALL FINISHED WITH SHORTIES---
9533- 60    .5     RTS
            *--------------------------------
            *      STRING POOL STROLL
            *--------------------------------
            SET.STRING.POOL.STROLL
9534- A6 74        LDX MEMSIZE+1 POINT FRESPC
9536- A5 73        LDA MEMSIZE   AT HIMEM
9538- 85 71        STA FRESPC    TO START
953A- 86 72        STX FRESPC+1  STROLL.
953C- 60           RTS
            *--------------------------------
            *   SEARCH STRING POOL FROM TOP TO BOTTOM
            *      FOR A NEGATIVE BYTE.
            *
            *      RETURN .CS. IF NEG BYTE FOUND,
            *             .CC. IF REACHED END OF POOL
            *--------------------------------
            FIND.NEXT.NEG.BYTE.IN.POOL
953D- A6 72        LDX FRESPC+1
953F- A4 71        LDY FRESPC
9541- A9 00        LDA #0       PAGE AT A TIME
9543- 85 71        STA FRESPC
9545- 98           TYA          IS IT ZERO?
9546- D0 07        BNE .2       NO!
9548- CA    .1     DEX          YES
9549- E4 70        CPX FRETOP+1 STILL IN POOL?
954B- 90 15        BCC .5       ...NO
954D- 86 72        STX FRESPC+1 DO NEXT PAGE
954F- 88    .2     DEY
9550- F0 06        BEQ .3
9552- B1 71        LDA (FRESPC),Y
9554- 10 F9        BPL .2       POS ASCII
9556- 30 04        BMI .4       NEG SO QUIT
9558- B1 71 .3     LDA (FRESPC),Y
955A- 10 EC        BPL .1 NEW PAGE
955C- E4 70 .4     CPX FRETOP+1
955E- D0 02        BNE .5       FRESPC>FRETOP
9560- C4 6F        CPY FRETOP   FOR CARRY FLAG
9562- 84 71 .5     STY FRESPC   FRESPC POINTS TO NEG ASC
9564- 60           RTS
            *--------------------------------
            *   SET UP SEARCH OF VAR TABLE
            *--------------------------------
            INITIATE.SEARCH
9565- A5 69        LDA VARTAB   START AT BEGINNING OF VARIABLES
9567- 85 19        STA INDEX
9569- A6 6A        LDX VARTAB+1
956B- 86 1A        STX INDEX+1
956D- A0 07        LDY #7       EACH VAR TAKES 7 BYTES
956F- 84 1B        STY OFFSET
9571- 60           RTS
            *--------------------------------
            *   FIND NEXT STRING VARIABLE
            *--------------------------------
            FIND.NEXT.STRING.VARIABLE
9572- A6 1A .1     LDX INDEX+1    SETUP SEARCH FOR NEXT STRING
9574- A5 19        LDA INDEX
9576- A4 1B        LDY OFFSET
9578- C0 07        CPY #7       STILL IN SIMPLE VARIABLES?
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

Single-Step *mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

Trace *mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

Background *mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

QUICKTRACE *allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

Two optional display formats *can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

QUICKTRACE *is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

QUICKTRACE *is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

QUICKTRACE *is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while* ***QUICKTRACE*** *is alive.*

QUICKTRACE *is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE ***$50***

Is a trademark of Anthro-Digital, Inc.

Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

```
957A- D0 18             BNE .4          ...NO, IN ARRAYS
957C- E4 6C             CPX ARYTAB+1 WE WERE, CHECK FURTHER...
957E- 90 04             BCC .2          ...YES, STILL SIMPLE
9580- C5 6B             CMP ARYTAB
9582- B0 0A             BCS .3          ...NO
9584- 20 E6 95  .2      JSR IS.THIS.A.STRING.VARIABLE
9587- B0 4C             BCS .8          ...STRING FOUND
9589- 20 D9 95          JSR NXTEL       NOT A STRING
958C- 90 E4             BCC .1          ...ALWAYS, TRY AGAIN
958E- 46 1B     .3      LSR OFFSET      SET OFFSET TO 3 NOW
9590- 85 1D             STA ARRAY.END
9592- 86 1E             STX ARRAY.END+1
9594- E4 1E     .4      CPX ARRAY.END+1 INSIDE AN ARRAY?
9596- 90 3D             BCC .8          ...YES
9598- C5 1D             CMP ARRAY.END
959A- 90 39             BCC .8
959C- E4 6E             CPX STREND+1 STILL IN VAR SPC?
959E- 90 05             BCC .5          ...YES
95A0- C5 6D             CMP STREND
95A2- 90 01             BCC .5          ...YES
95A4- 60                RTS             CARRY SET WHEN THRU VAR SPC
                *---SET UP A NEW ARRAY-----------
95A5- A0 02     .5      LDY #2
95A7- 18                CLC
95A8- B1 19             LDA (INDEX),Y
95AA- 65 19             ADC INDEX
95AC- 85 1D             STA ARRAY.END   POINTER TO
95AE- C8                INY             NEXT ARRAY
95AF- B1 19             LDA (INDEX),Y
95B1- 65 1A             ADC INDEX+1
95B3- 85 1E             STA ARRAY.END+1
95B5- 20 E6 95          JSR IS.THIS.A.STRING.VARIABLE IS THIS A STR?
95B8- B0 0A             BCS .6          ...YES
95BA- A5 1D             LDA ARRAY.END
95BC- 85 19             STA INDEX       NO
95BE- A6 1E             LDX ARRAY.END+1
95C0- 86 1A             STX INDEX+1
95C2- D0 D0             BNE .4          ...ALWAYS
                *---FOUND STRING ARRAY-----------
95C4- A0 04     .6      LDY #4          POINT AT
95C6- B1 19             LDA (INDEX),Y     #DIMENSIONS OF ARRAY
95C8- 0A                ASL             *2
95C9- 69 05             ADC #5
95CB- 65 19             ADC INDEX       POINT INDEX TO
95CD- 85 19             STA INDEX       FIRST ELEMENT
95CF- 90 02             BCC .7          OF NEW ARRAY
95D1- E6 1A             INC INDEX+1
95D3- A6 1A     .7      LDX INDEX+1
                *
95D5- 85 9B     .8      STA LOWTR       LOWTR->STR DESCRIPTOR
95D7- 86 9C             STX LOWTR+1
                *---NEXT VARIABLE----------------
95D9- 18        NXTEL   CLC
95DA- A5 1B             LDA OFFSET      POINT INDEX TO
95DC- 65 19             ADC INDEX       NEXT VAR OR ELEMENT
95DE- 85 19             STA INDEX
95E0- 90 03             BCC .1
95E2- E6 1A             INC INDEX+1
95E4- 18                CLC
95E5- 60        .1      RTS             STR FOUND,CARRY CLEAR
                *--------------------------------
                * SUBROUTINE STRING CHECK
                *--------------------------------
                IS.THIS.A.STRING.VARIABLE
95E6- A0 00             LDY #0
95E8- 18                CLC             INCASE NOT STR
95E9- B1 19             LDA (INDEX),Y
95EB- 30 0D             BMI .2          ...NOT STRING
95ED- C8                INY
95EE- B1 19             LDA (INDEX),Y
95F0- 10 08             BPL .2          ...NOT STRING
95F2- A9 02             LDA #2          POINT PAST STR NAME
95F4- 65 19             ADC INDEX
95F6- 90 01             BCC .1          ...STRING
95F8- E8                INX             INDEX+1
95F9- 38        .1      SEC             CARRY SET IF STR FOUND
95FA- 60        .2      RTS
                *--------------------------------
```

Changing VERIFY to DISPLAY................Bob Sander-Cederlof

In the July 1982 issue of AAL we showed how to make a text file display command inside DOS. Bob Bragner added 80-column output to it in the July 1983 issue. The Dec 1983 InCider printed an article by William G. Wright about a DOS modification that provided text file display without removing any previous features.

Wright's patches modify the VERIFY command so that as sectors are being verified, if the file is a text file, they are displayed on the screen or printer. If there are any $00 bytes in a sector, they are merely skipped over, so his patches will handle some random access files, as well as sequential. Non-text files are still verified in the normal manner.

I was prompted by his article to write up another little program. This one will hook into the VERIFY processor in the file manager when you BRUN the program. Later, 30BG from the monitor or CALL 779 from Applesoft will dis-install the patch. My patch modifies VERIFY so that as each sector of a file is verified it is displayed in hexadecimal on the screen or a printer. I do not distinguish between text and non-text files, although it would be a simple matter to do so. As with Wright's patches, random access files may also be displayed, up to the first hole in the track/sector list.

The creative among you will want to add many bells and whistles to my little program. Perhaps 80-column display, showing an entire sector at a time rather than half a sector. Perhaps display of the bytes in both hex and ASCII on the same line. Perhaps the ability to scan back and forth through a file, using the arrow keys. All these are possible, and not too difficult. Have fun!

```
                   1000 *SAVE S.DISPLAY FILE
                   1010 *--------------------------------
                   1020 *      PATCH DOS TO CHANGE VERIFY
                   1030 *      INTO DISPLAY
                   1040 *--------------------------------
F941-              1050 MON.PRNTAX .EQ $F941
F94A-              1060 MON.PRBL2  .EQ $F94A
FD8E-              1070 MON.CROUT  .EQ $FD8E
FDDA-              1080 MON.PRBYTE .EQ $FDDA
FDED-              1090 MON.COUT   .EQ $FDED
                   1100 *--------------------------------
                   1110        .OR $300
                   1120 *--------------------------------
0300- A9 16        1130 PATCH  LDA #DISPLAY      HOOK INTO VERIFY COMMAND
0302- 8D 1C AD     1140        STA $AD1C
0305- A9 03        1150        LDA /DISPLAY
0307- 8D 1D AD     1160        STA $AD1D
030A- 60           1170        RTS
                   1180 *--------------------------------
                   1190 UNPATCH
030B- A9 B6        1200        LDA #$B0B6        RESTORE NORMAL VERIFY
030D- 8D 1C AD     1210        STA $AD1C
0310- A9 B0        1220        LDA /$B0B6
0312- 8D 1D AD     1230        STA $AD1D
0315- 60           1240        RTS
                   1250 *--------------------------------
                   1260 DISPLAY
0316- 20 8E FD     1270        JSR MON.CROUT     START SECTOR WITH <RET>
0319- 20 B6 B0     1280        JSR $B0B6         READ NEXT SECTOR
031C- B0 09        1290        BCS .1            END OF FILE
031E- A0 00        1300        LDY #0            DISPLAY FIRST HALF SECTOR
```

```
0320- 20 28 03              JSR SHOW
0323- 20 28 03              JSR SHOW          DISPLAY SECOND HALF
0326- 18                    CLC               SIGNAL NOT END OF FILE
0327- 60          .1        RTS
                  *-------------------------------
0328- AD E5 B5    SHOW      LDA $B5E5     DISPLAY SECTOR POSITION
032B- AE E4 B5              LDX $B5E4
032E- 20 41 F9              JSR MON.PRNTAX
0331- A9 10                 LDA #16       16 LINES PER BLOCK
0333- 8D 65 03              STA LCNT
0336- D0 05                 BNE .2        ...ALWAYS
0338- A2 04       .1        LDX #4        PRINT 4 BLANKS
033A- 20 4A F9              JSR MON.PRBL2     SO COLUMNS LOOK NEATER
033D- A9 08       .2        LDA #8        8 BYTES PER LINE
033F- 8D 64 03              STA BCNT
0342- 98                    TYA           PRINT BYTE COUNT
0343- 20 DA FD              JSR MON.PRBYTE
0346- A9 AD                 LDA #"-"      PRINT "-"
0348- 20 ED FD              JSR MON.COUT
034B- A9 A0       .3        LDA #" "      PRINT " "
034D- 20 ED FD              JSR MON.COUT
0350- B1 42                 LDA ($42),Y   NEXT BYTE FROM FILE
0352- C8                    INY
0353- 20 DA FD              JSR MON.PRBYTE
0356- CE 64 03              DEC BCNT
0359- D0 F0                 BNE .3        MORE TO THIS LINE
035B- 20 8E FD              JSR MON.CROUT     NEXT LINE
035E- CE 65 03              DEC LCNT
0361- D0 D5                 BNE .1
0363- 60                    RTS
                  *-------------------------------
0364-             BCNT     .BS 1
0365-             LCNT     .BS 1
                  *-------------------------------
```

Changing Tab Stops in the 68000 Cross Assembler................

Bob Sander-Cederlof

The procedure as described in the S-C Macro Assembler manual works for the 6502 version and for all the cross assemblers except the 68000 cross assembler. The procedure described in Appendix D will not work because the 68000 cross assembler uses both banks of memory at $D000-DFFF. In order to be certain the correct one is switched on, the command interpreter keeps using the selection soft switches. The result is that the bank stays write-protected, and no patches ever get installed.

Of course, there is a simple way around the problem. Here is how to change the tab stops in the 68000 Cross Assembler:

First, boot the cross assembler disk and select option 2, loading the language card version at $D000.

```
:BLOAD S-C.ASM.MACRO.68000.LC
:MNTR
*AA60.AA61
*AA60- xx yy   (probably C6 27)
*D010.D014
D010- 0E 16 1B 20 00
*C083 C083 D010:7 10 1B 2B  (or whatever values you like)
C083- zz
C083- zz
*D010.D014
D010- 07 10 1B 2B 00
*BSAVE S-C.ASM.MACRO.68000.LC,A$D000,L$yyxx
*3D0G
:     that's it!
```

Similar methods apply to the other customizing patches mentioned in Appendix D.

S-C Macro and GPLE.LC on the //e..................Bob Bragner
Istanbul, Turkey

I've long been bothered by the way loading the S-C Macro Assembler wipes out GPLE (Neil Konzen's Global Program Line Editor) when you go from Applesoft to Assembler. It shouldn't happen, because GPLE resides in the alternate bank at $D000, not used by Macro. I've also been unhappy that the //e version of S-C Macro Assembler doesn't have the automatic line of dashes provided by <esc> L after a line number when you are in 80-column mode.

Just the other day I discovered by accident that all is not lost. If things are done in just the right sequence both of my peeves vanish.

First load up the S-C Macro Assembler into the $D000 area. Then enter Applesoft by typing the FP command, and BRUN GPLE.LC. Initialize the 80-column card with ctrl-V and enter the assembler by typing the INT command. This leaves GPLE connected so that the assembler sees the <esc> L command. Try it by typing a line number and <esc> L.

It also allows the assembler to see <esc> L to turn a catalog line into a LOAD command, but due to the way the word LOAD is poked onto the screen you get L O A D which clobbers the file name. (I never use the automatic load anyway, so this does not bother me.)

RESET will partially disable GPLE.LC, but you can restore it by typing the & command from Applesoft. If you want RESET to NOT molest GPLE, change the reset vector to $B6B3. You can do this from the monitor with "3F2:B3 B6 13", or from S-C Macro with "RST $B6B3".

I don't know why this all works, but I think it has something to do with the way the 80-column card initializes itself by copying the //e's monitor ROM into the $F800-FFFF space of RAM.

By the way, GPLE uses some patch space inside DOS 3.3 which is also used by the fast DOS text file I/O patch, so beware of mixing them.

Will Rockwell 65C02's work in an old Apple..........Bob Stout

Not unless you have the 2 MHz part. For some reason the timing is too tight and slightly different to use a 1MHz 65C02, unless you have an Apple //e. The 2 MHz chips WILL work in Apple II and II Plus.

Redundancy in Tables for Faster Lookups....Bob Sander-Cederlof

When speed is the main objective, you can sometimes use table lookups to great advantage. You trade program size for speed.

Here is an easy example. Suppose I want to convert the two nybbles of a byte to ASCII characters. I can do it all with code, like this:

```
CONVERT
       PHA                Save original byte
       LSR                Position first nybble
       LSR
       LSR
       LSR
       JSR MAKE.ASCII
       STA XXX
       PLA                Original byte
       AND #$0F           Isolate second nybble
       JSR MAKE.ASCII
       STA XXX+1
       RTS

MAKE.ASCII
       ORA #$B0           Make B0...BF
       CMP #$BA
       BCC .1             It is 0-9
       ADC #6             Make A-F codes
.1     RTS
```

That takes 30 bytes, and 75-77 cycles including a JSR CONVERT to call it. Actually 75 cycles if both nybbles are 0-9, 77 cycles if they both are A-F, and 76 cycles if there is one of each. If I move the code from MAKE.ASCII in-line, it saves 24 cycles (two JSRs, two RTSs), and only lengthens the program by one byte.

Or I can do a table lookup by substituting these two lines for both JSR MAKE.ASCII lines above:

```
       TAX
       LDA ASCII.TABLE,X
```

and making a little table like this:

```
ASCII.TABLE .AS -/0123456789ABCDEF/
```

In this form, the program takes 49 cycles, and uses a total of 39 bytes including the table. Perhaps it could be an advantage that the # of cycles is always constant, regardless of the value being converted.

You can make it even faster by using two whole pages for table space, like this:

```
CONVERT
       TAX
       LDA HI.TABLE,X
```

```
       STA XXX
       LDA LO.TABLE,X
       STA XXX+1
       RTS

HI.TABLE
       .AS -/0000000000000000/
       .AS -/1111111111111111/
       .
       .
       .AS -/FFFFFFFFFFFFFFFF/

LO.TABLE
       .AS -/0123456789ABCDEF/
       .AS -/0123456789ABCDEF/
       .
       .
       .AS -/0123456789ABCDEF/
```

The program itself is 14 bytes long, but there are 512 bytes of tables. The conversion, including JSR and RTS, now takes only 30 cycles. And since the program is now so short, it would probably get placed in line, saving the JSR-RTS, converting in only 18 cycles. And if the in-line routine already had the nybble in the X-reg, whack off another two cycles.

The redundancy in the tables gives a huge speed increase.

I have been tearing into the super fast copy utility that comes with Locksmith 5.0, and I discovered some of these redundancy tricks in their disk I/O tables. For example, the table for converting a six-bit value into a disk-code normally takes 64 bytes. The table looks like this:

```
TABLE  .HS 96979A9B9D9E9FA6
       .
       .
       .HS F7F9FAFBFCFDFEFF
```

Code to access the table might look like this:

```
       LDA BUFFER,X
       AND #$3F        Mask to 6 bits
       TAY
       LDA TABLE,Y
```

When you are writing to a disk, every single cycle counts. Therefore, it is pleasant to discover redundant tables. By making four copies of the table, using 256 bytes rather than 64, we no longer need to strip off the first two bits. The code can be shortened to this:

```
       LDY BUFFER,X
       LDA TABLE,Y
```

It only saves 3 cycles, but those three cycles can and do make the whole difference in the fast copy program. That is part of Locksmith's secret to reading a whole disk into RAM in only 8 seconds.

Speaking of Locksmith........................Warren R. Johnson

Did you know that Locksmith 5.0 can nearly be copied by plain old COPYA? Or with its own fast backup copier? All but the last few tracks copy, and they may not be necessary.

The only problem is, the resulting copy will not boot until you make a small patch using some sort of disk ZAP utility. (You can use Omega's Inspector/Watson team, Bag of Tricks, Disk Fixer, CIA, for example.) Patch Track-0F Sector-0E Byte-6F: change it from 6C to 0F. [Editor's note: in my copy, Locksmith had C6 in that byte rather than 6C. And I have not tried the resulting disk to see if all functions work.]

I have modified my Apple a little to make my life easier. I have 2732's in the motherboard ROM sockets, with bank switch selection. Applesoft is in one bank, and a modified version of Applesoft in the other. My modifications include replacing the old cassette commands (LOAD/SAVE/SHLOAD etc.) for an INWAT command. INWAT downloads the Inspector and Watson from some expansion chassis ROM boards.

Lancaster's OBJ.APWRT][F.................Bob Sander-Cederlof

You may have noticed a little ad in the last few issues for an obscure title, "OBJ.APWRT][F". Don Lancaster, author of such favorites as Enhancing the Apple, Incredible Secret Money Machine, Micro Cookbook, etc., has torn into Applewriter //e. After a thorough analysis, he completely documented it, in the style of Beneath Apple DOS. The results, or at least part of them, will be chapter 12 in volume 2 of Enhancing.

He sent me a pre-print to look at and make comments about. My main comment is WOW! It doesn't matter if you like Applewriter or not. It doesn't even matter if you have never seen Applewriter. You still can learn a tremendous amount by reading through Don's text and comments. Of course it is better if you DO have Applewriter //e, because he tells you how to make some great customizing modifications.

You can get it all on disk for only $29.95. Actually, it is not on "disk" ... it is on SIX disk sides, jam-packed full. Don even throws in a free book for good measure.

You can order OBJ.APWRT][F directly from Synergetics, or from S-C Software.

About Disk Drive Pressure Pads............Bob Sander-Cederlof

After you have used your disk drive for six months or so, it will probably develop a scary noise or two. I know mine have.

My oldest drive is serial 1901 (the Shugart mechanics inside the box have a number somewhere in the low 400's). Every once and a while it will make the most dangerous sounding noise you ever heard, something like dragging rusty chains across the road. I have read in various magazines and newsletters that these noises are almost always caused by a dirty pressure pad.

The pressure pad rides on the top surface of the disk, pressing the disk surface down against the recording head. It is a 1/8 inch circle of felt glued to a slightly larger plastic stud. The shaft of the stud is split and tapered, so it will fit through a hole and lock in place. You can easily remove the pad and stud by pressing on the split end.

But where do you get new ones? Maybe at a computer store, but they sure don't keep them on display. I decided to try a little home maintenance, and it worked. I gently scraped the felt surface with the blade of my pocket knife, and all the old caked oxide turned to powder and fell off. Then I rubbed the oxide on a piece of paper, to smooth out the felt. After putting the drive all back together, it ran quietly.

It worked so well, I performed the operation on two more drives. And surprisingly, one drive which had been giving lots of errors was working accurately again.

A few other disk maintenance tips:

> One particularly noisy drive a few years ago had loose screws trying to hold the drive motor down. A few wrist twists and all was well.
>
> If a drive can read, but writes garbage, it is probably the 74LS125 on the analog board inside the drive. Replace that chip for 25 cents or so, and you have saved $60 in repair bills.

Will ProDOS Work on a Franklin?....................Bob Stout

If you try to boot up ProDOS on a Franklin, it probably will fail. The ProDOS boot routine checks to see if you are in a genuine Apple monitor ROM. However, you can make it work.

Start the boot procedure; when meaningful action appears to have ceased, press the RESET switch. Get into the monitor and type 2647:EA EA and 2000G. Voila!

Rod's Color Pattern in 6502 Code............Charles H. Putney

When I read the January AAL with Bob Urschel's article about running Rod's Color Pattern on the QWERTY 68000 board, it sounded like a challenge. You may remember I like speed challenges, at least inside computers.

Fifty times faster than Basic didn't sound too fast, so I checked a simple loop to see if it might be possible to save the dignity of the 6502. It did look possible, at least by using tricky table-driven code.

So, I wrote some more code and it looked like 8.0 seconds per loop. This clocks out at 55 times faster than Integer BASIC, but I didn't have the internal calculation for the color value exactly like the original.

I finally decided to use a table lookup for the color calculation. Now the problem was how to create all those data statements. I thought about using some macros, but the calculations are too involved. I wrote an Applesoft program to generate the lines of code for the assembler, and then EXECed them into my source code. I finally got all the bugs out and timed it.

The table-driven version performs a main loop every 6.2 seconds, compared to 446 seconds per loop for the Integer BASIC version. That is nearly 72 times faster.

Well, my only worry now is that Bob Urschel made an error in his timing, and his really runs 200 times faster. If not, we have saved face for the venerable 6502.

Of course, we did use a little more memory. But that is frequently a trade-off worth making in important programs.

For comparison purposes, here is the Integer BASIC program again:

```
10 GR
20 FOR W = 3 TO 50
30 FOR I = 1 TO 19
40 FOR J = 0 TO 19
50 K = I+J
60 COLOR = J*3 / (I+3) + I*W/12
70 PLOT I,K: PLOT K,I: PLOT 40-I,40-K:PLOT40-K,40-I
80 PLOT K,40-I: PLOT 40-I,K: PLOT I,40-K: PLOT 40-K,I
90 NEXT J: NEXT I: NEXT 2
100 GO TO 20
```

My program to generate the data tables includes similar logic. I broke the tables into two planes, rather than storing one data table 48*19*20 = 18,240 bytes long. One plane computes J*3/(I+3), and includes 380 bytes. The other computes I*W/12 and includes 48*19=912 bytes. My table lookup routine uses I and J to index into the first plane, and I and W into the second. Then the two values are added together. Pretty tricky!

```
100  REM BUILD TABLES FOR ROD'S COLOR
110 D$ =  CHR$ (4)
120  PRINT D$"OPEN TTT": PRINT D$"WRITE TTT"
130  GOSUB 200
140  PRINT D$"CLOSE"
150  END
200  FOR I = 1 TO 19
210  PRINT "WI" LEFT$ ( STR$ (I) + " ",2)" .HS ";
220  FOR W = 3 TO 26: GOSUB 500: NEXT : PRINT
230  PRINT "      .HS ";
240  FOR W = 27 TO 50: GOSUB 500: NEXT : PRINT
250  NEXT I
300  FOR I = 1 TO 19
310  PRINT "JI" LEFT$ ( STR$ (I) + " ",2)" .HS ";
320  FOR J = 0 TO 19
330 C =  INT (J * 3 / (I + 3)):C = C -  INT (C / 16) * 16: IF C > 9 THEN
     C = C + 7
340  PRINT "0" CHR$ (C + 48);
350  NEXT : PRINT
360  NEXT I
370  RETURN
500 C =  INT (I * W / 12):C = C -  INT (C / 16) * 16: IF C > 9 THEN C = C
      + 7
510  PRINT "0" CHR$ (C + 48);
520  RETURN
```

I believe in letting computers work for me, so I had to use some macros to simplify typing in all the code for those eight plot statements. I wrote a PLOT macro, but then I noticed that there was some redundant code that way. By rearranging the order of the PLOT statements, I can separate the y-setup from the x-setup and plot. That way the base address does not get re-calculated as often, saving more time. Here is my program:

```
                 *SAVE S.PUTNEY'S COLOR
                        .OR $6000
                        .TF B.PUTNEY
                 *-------------------------------
                 *
                 *      FAST ROD'S COLOR PATTERN
                 *
                 *      CHARLES H. PUTNEY
                 *      18 QUINNS ROAD
                 *      SHANKILL
                 *      CO. DUBLIN
                 *      IRELAND
                 *
                 *-------------------------------
                 *
                 *      PAGE ZERO ADDRESSES
                 *
00EE-            INVI   .EQ $EE       VARIABLE 40 - I
00EF-            INVK   .EQ $EF       VARIABLE 40 - K
00F9-            POINTR .EQ $F9       LORES PAGE POINTER (TWO BYTES)
00FB-            I      .EQ $FB       VARIABLE I
00FC-            J      .EQ $FC       VARIABLE J
00FD-            K      .EQ $FD       VARIABLE K
00FE-            W      .EQ $FE       VARIABLE W
0007-            COLOR1 .EQ $07       HALF OF COLOR FORMULA
                 *-------------------------------
0008-            COLOR  .EQ $08,09
0008-            COLEVN .EQ $08       EVEN ROW COLOR
0009-            COLODD .EQ $09       ODD ROW COLOR
000A-            MASK   .EQ $0A,0B
000A-            MSKODD .EQ $0A
000B-            MSKEVN .EQ $0B
                 *
```

```
                     1340 *--------------------------------
                     1350 *
                     1360 *       ADDRESS TABLE
                     1370 *
00F0-                1380 ODDMSK .EQ $F0      MASK FOR ELIMINATING UPPER BLOCK
000F-                1390 EVNMSK .EQ $0F      MASK FOR ELIMINATING LOWER BLOCK
FB40-                1400 GRAPH  .EQ $FB40    ENABLE LO RES GRAPHICS
                     1410 *--------------------------------
                     1420 *   MACRO DEFINITIONS
                     1430 *--------------------------------
                     1440        .MA PLY       PLY ]1
                     1450        LDY ]1        Y-COORD
                     1460        LDA LORESL,Y
                     1470        STA POINTR
                     1480        LDA LORESH,Y
                     1490        STA POINTR+1
                     1500        TYA
                     1510        AND #1        GET LSB
                     1520        TAX
                     1530        .EM
                     1540 *--------------------------------
                     1550        .MA PLX       PLX ]1
                     1560        LDY ]1        X-COORD
                     1570        LDA (POINTR),Y GET THE PAGE BYTE
                     1580        AND MASK,X
                     1590        ORA COLOR,X
                     1600        STA (POINTR),Y PUT IT BACK
                     1610        .EM
                     1620 *--------------------------------
                     1630        .MA NEXT      NEXT VAR,LIMIT+1
                     1640        INC ]1        INCREMENT ]1 VARIABLE
                     1650        LDA ]1        TEST IF ]1=]2 YET
                     1660        CMP #]2
                     1670        BCS :1        YES, LEAVE LOOP
                     1680        JMP NEXT.]1   NO, KEEP LOOPING
                     1690 :1
                     1700        .EM
                     1710 *--------------------------------
                     1720        .MA GET       GET FORMULA,INDEX
                     1730        LDY I
                     1740        LDA ]1L-1,Y
                     1750        STA POINTR
                     1760        LDA ]1H-1,Y
                     1770        STA POINTR+1
                     1780        LDY ]2
                     1790        LDA (POINTR),Y
                     1800        .EM
                     1810 *--------------------------------
                     1820 *
                     1830 *
                     1840 *      MAIN LOOP
                     1850 *
6000- AD 56 C0       1860 ROD    LDA $C056     SET LORES GRAPHICS ON
6003- AD 53 C0       1870        LDA $C053     MIXED MODE
6006- 20 40 FB       1880        JSR GRAPH
6009- A9 F0          1890        LDA #ODDMSK
600B- 85 0A          1900        STA MSKODD
600D- A9 0F          1910        LDA #EVNMSK
600F- 85 0B          1920        STA MSKEVN
                     1930 *--------------------------------
                     1940 BIG.LOOP
6011- 20 E2 FB       1950        JSR $FBE2     *** TESTING BEEP ***
6014- A9 00          1960        LDA #0        FOR W=3 TO 50 (0...47)
6016- 85 FE          1970        STA W
                     1980 *---NEXT W COMES HERE------------
6018- A9 01          1990 NEXT.W LDA #1        FOR I=1 TO 19
601A- 85 FB          2000        STA I
601C- A9 27          2010        LDA #39
601E- 85 EE          2020        STA INVI
6020- AD 30 C0       2030        LDA $C030     JUST FOR AUDIBLE FEEDBACK
                     2040 *---NEXT I COMES HERE------------
6023- A5 FB          2050 NEXT.I LDA I         SET UP K = I+J
6025- 85 FD          2060        STA K
6027- A5 EE          2070        LDA INVI
6029- 85 EF          2080        STA INVK
602B-                2090        >GET FORM1,W
603B- 85 07          2100        STA COLOR1    SAVE IT FOR INNER LOOP
603D- A9 00          2110        LDA #0        FOR J=0 TO 19
603F- 85 FC          2120        STA J
```

```
                  2130 *---NEXT J COMES HERE-----------
6041-             2140 NEXT.J >GET FORM2,J
6051- 18          2150        CLC           ADD THE FORMULAS
6052- 65 07       2160        ADC COLOR1    ACC = J*3/(I+3)+I*W/12
6054- 29 0F       2170        AND #$0F      MASK OFF TOP
6056- 85 08       2180        STA COLEVN    EVEN COLOR
6058- 0A          2190        ASL           SHIFT 4 BITS
6059- 0A          2200        ASL
605A- 0A          2210        ASL
605B- 0A          2220        ASL
605C- 85 09       2230        STA COLODD    ODD COLOR
                  2240 *-------------------------------
605E-             2250        >PLY I
606E-             2260        >PLX K
6078-             2270        >PLX INVK
                  2280 *-------------------------------
6082-             2290        >PLY INVI
6092-             2300        >PLX K
609C-             2310        >PLX INVK
                  2320 *-------------------------------
60A6-             2330        >PLY K
60B6-             2340        >PLX I
60C0-             2350        >PLX INVI
                  2360 *-------------------------------
60CA-             2370        >PLY INVK
60DA-             2380        >PLX I
60E4-             2390        >PLX INVI
                  2400 *-------------------------------
60EE- E6 FD       2410        INC K
60F0- C6 EF       2420        DEC INVK
60F2-             2430        >NEXT J,20
                  2440 *-------------------------------
60FD- C6 EE       2450        DEC INVI
60FF-             2460        >NEXT I,20
                  2470 *-------------------------------
610A-             2480        >NEXT W,48
                  2490 *-------------------------------
6115- AD 00 C0    2500        LDA $C000     ANY KEY PRESSED?
6118- 30 03       2510        BMI ROD4      YES
611A- 4C 11 60    2520        JMP BIG.LOOP    NO, KEEP LOOPING
611D- 8D 10 C0    2530 ROD4   STA $C010
6120- 60          2540        RTS
```

```
2550 *-----------------------------
2560 *
2570 *        LORES GRAPHICS PAGE
2580 *        BASE ADDRESSES (40 COL)
2590 *
2600 LORESL .HS 0000808000008080
2610        .HS 0000808000008080
2620        .HS 2828A8A82828A8A8
2630        .HS 2828A8A82828A8A8
2640        .HS 5050D0D05050D0D0
2650 LORESH .HS 0404040405050505
2660        .HS 0606060607070707
2670        .HS 0404040405050505
2680        .HS 0606060607070707
2690        .HS 0404040405050505
2700 *
2710 *-----------------------------
2720 *
2730 *        TABLE FOR I*W/12
2740 *
2750 FORM1L .DA #WI1,#WI2,#WI3,#WI4,#WI5,#WI6,#WI7
2760        .DA #WI8,#WI9,#WI10,#WI11,#WI12,#WI13,#WI14
2770        .DA #WI15,#WI16,#WI17,#WI18,#WI19
2780 FORM1H .DA /WI1,/WI2,/WI3,/WI4,/WI5,/WI6,/WI7
2790        .DA /WI8,/WI9,/WI10,/WI11,/WI12,/WI13,/WI14
2800        .DA /WI15,/WI16,/WI17,/WI18,/WI19
```

```
*--------------------------------
WI1   .HS 000000000000000000010101010101010101010101020202
      .HS 020202020202020202020303030303030303030303040404
WI2   .HS 000000010101010101020202020202020303030303040404
      .HS 040404050505050505060606060606070707070707080808
WI3   .HS 000101010102020202030303030404040405050505060606
      .HS 060707070707080808080909090A0A0A0A0B0B0B0B0C0C0C
WI4   .HS 010101020202030303040404050505060606070707080808
      .HS 0909090A0A0A0B0B0B0C0C0C0D0D0D0E0E0E0F0F0F000000
WI5   .HS 010102020203030404050505060607070708080909 0A0A0A
      .HS 0B0B0C0C0C0D0D0E0E0F0F0F000001010102020303040404
WI6   .HS 010202030304040505060607070808090 90A0A0B0B0C0C0D
      .HS 0D0E0E0F0F000001010202030304040505060607070808 09
WI7   .HS 0102020304040505060707080809090A0B0B0C0C0D0E0E0F
      .HS 0F000001020203030405050606070708090 90A0A0B0C0C0D
WI8   .HS 020203040405060607080809 0A0A0B0C0C0D0E0E0F000001
      .HS 020203040405060607080809 0A0A0B0C0C0D0E0E0F000001
WI9   .HS 0203030405060607080909 0A0B0C0C0D0E0F0F0001020203
      .HS 04050506070808090A0B0B0C0D0E0E0F0C01010203040405
WI10  .HS 020304050506070809 0A0A0B0C0D0E0F0F00010203040405
      .HS 0607080909 0A0B0C0D0E0E0F000102030304050607080809
WI11  .HS 02030405060708090A0B0B0C0D0E0F000102030405060607
      .HS 08090A0B0C0D0E0F0001010203040506070809 0A0B0C0C0D
WI12  .HS 030405060708090A0B0C0D0E0F000102030405060708090A
      .HS 0B0C0D0E0F000102030405060708090A0B0C0D0E0F000102
WI13  .HS 030405060708090A0B0D0E0F00010203040506070 80A0B0C
      .HS 0D0E0F00010203040507080 90A0B0C0D0E0F000102040506
WI14  .HS 0304050708090A0B0C0E0F0001020305060708090A0C0D0E
      .HS 0F000103040506070 80A0B0C0D0E0F0102030405060809 0A
WI15  .HS 03050607080A0B0C0D0F0001020405060709 0A0B0C0E0F00
      .HS 010304050608090A0B0D0E0F000203040507080 90A0C0D0E
WI16  .HS 04050608090A0C0D0E000102040506 08090A0C0D0E000102
      .HS 04050608090A0C0D0E000102040506 08090A0C0D0E000102
WI17  .HS 0405070809 0B0C0E0F010203050608090A0C0D0F00020304
      .HS 0607090A0B0D0E000103040507080A0B0C0E0F0102040506
WI18  .HS 04060709 0A0C0D0F0002030506 08090B0C0E0F0102040507
      .HS 080A0B0D0E00010304060709 0A0C0D0F00020305060809 0B
WI19  .HS 0406070 90B0C0E0F0103040607090A0C0E0F01020406070 9
      .HS 0A0C0D0F01020405070 90A0C0D0F0002040507080A0C0D0F
*--------------------------------
*
*       TABLE FOR J*3/(I+3)
*
FORM2L .DA #JI1,#JI2,#JI3,#JI4,#JI5,#JI6,#JI7
       .DA #JI8,#JI9,#JI10,#JI11,#JI12,#JI13,#JI14
       .DA #JI15,#JI16,#JI17,#JI18,#JI19
FORM2H .DA /JI1,/JI2,/JI3,/JI4,/JI5,/JI6,/JI7
       .DA /JI8,/JI9,/JI10,/JI11,/JI12,/JI13,/JI14
       .DA /JI15,/JI16,/JI17,/JI18,/JI19
*--------------------------------
JI1   .HS 00000102030304050606070809090A0B0C0C0D0E
JI2   .HS 0000010102030304040506060707080909 0A0A0B
JI3   .HS 0000010102020303040405050606070708080909
JI4   .HS 0000000101020203030304040505060606070708
JI5   .HS 0000000101010202030303040404050506060607
JI6   .HS 0000000101010202020303030404040505050606
JI7   .HS 0000000001010102020203030303040404050505
JI8   .HS 0000000001010101020202030303030404040405
JI9   .HS 0000000001010101020202020303030304040404
JI10  .HS 0000000000010101010202020203030303030404
JI11  .HS 0000000000010101010102020202030303030304
JI12  .HS 0000000000010101010102020202020303030303
JI13  .HS 0000000000000101010101020202020202030303
JI14  .HS 0000000000000101010101010202020202030303
JI15  .HS 0000000000000101010101010202020202020303
JI16  .HS 0000000000000001010101010102020202020203
JI17  .HS 0000000000000000010101010101010202020202
JI18  .HS 0000000000000000010101010101010202020202
JI19  .HS 0000000000000000000101010101010102020202
*--------------------------------
```

Will ProDOS Really Fly?...................Bob Sander-Cederlof

ProDOS appears to have been eclipsed by MacIntosh. The major software houses are probably putting their main effort into Mac.

ARTSCI has announced a ProDOS version of their MagiCalc spreadsheet program. Owners of the DOS 3.3 version may upgrade for $40, new customers pay $149.95. The only differences claimed are faster disk I/O and ability to edit the printer setup string. Nice, but $40 is a lot. And the spreadsheet files would no longer be accessible to DOS-based utilities.

ARTSCI will also send you their ProDOS catalog sorter program, complete with BASIC.SYSTEM, CONVERT, FILER, and the ProDOS image for only $24.95. Apple will reputedly be selling ProDOS with a user's manual and some tutorial files in addition to the files on ARTSCI's disk, but price and date are still unclear. (You get them free with a new disk drive.)

Practical Peripherals has announced a new clock card which is ProDOS compatible. Their design is apparently an upgrade of Superclock II (by Jeff Mazur, Westside Electronics). ProDOS was designed around Thunderclock, so other clocks must either emulate one of the Thunderclock modes or patch ProDOS during the boot process. Applied Engineering's new Timemaster II emulates Thunderclock and several others, so it is fully ProDOS compatible.

According to Don Lancaster, Applewriter //e has been written so that changing to ProDOS would be easy. Therefore we might expect a ProDOS-based version of this popular word processor to be announced soon. Or maybe they won't bother to announce it.

Meanwhile, I know of at least two people with plans to integrate the faster RWTS ProDOS uses into their enhanced DOS 3.3 packages. Have you seen the latest ads for David-DOS? Dave Weston compares the speeds of his fast DOS with DOS 3.3 and Pro-DOS. Guess what ... Pro-DOS doesn't win.

Unless you MUST have file compatibility with Apple /// SOS; or you MUST have hard hard-disk support for very large files; or you MUST have a hierarchical file directory; then stick with DOS 3.3, enhanced by Dave, or Bill Basham, or Art Schumer, or others. And if you MUST have at least 32K of program space with Applesoft; or you MUST have Integer BASIC support; or you MUST have compatibility with hundreds of existing software products; then stick with DOS 3.3.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).